Machine Learning Course

Introduction to Deep Learning

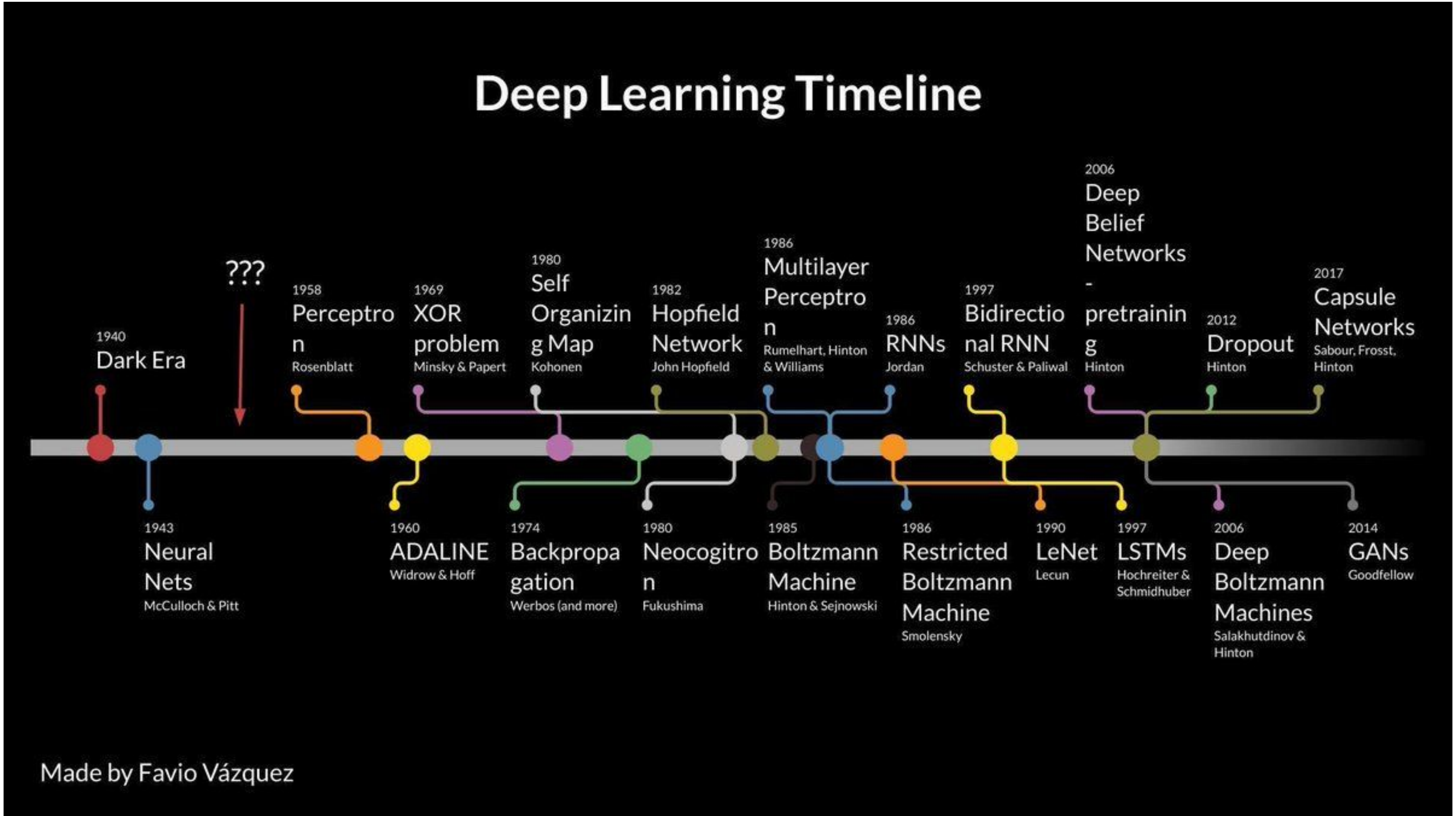
Neychev Radoslav
ML Instructor, BigData Team
Research Scientist, MIPT

03.09.2018, Moscow, Russia

Outline

1. Neural Networks in different areas. Historical overview.
2. Neural Networks basis. Backpropagation, chain rule.
3. Layers, activations, optimizers.
4. More layers and intuition.
5. Embeddings
6. Recurrent Neural Networks for signal and text processing.
7. RNNs in the wild (names generation from scratch).

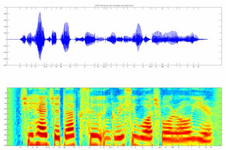
Materials: http://bit.ly/ml4megafon_august18_public



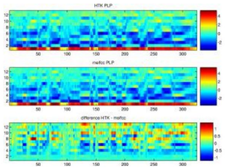
Deep Learning Timeline
1940
Dark Era
???
1943
Neural Nets
McCulloch & Pitt
1958
Perceptron
Rosenblatt
1960
ADALINE
Widrow & Hoff
1969
XOR problem
Minsky & Papert
1974
Backpropagation
Werbos (and more)
1980
Self Organizing Map
Kohonen
1980
Neocogitron
Fukushima
1982
Hopfield Network
John Hopfield
1985
Boltzmann Machine
Hinton & Sejnowski
1986
Multilayer Perceptron
Rumelhart, Hinton & Williams
1986
RNNs
Jordan
1986
Restricted Boltzmann Machine
Smolensky
1990
LeNet
Lecun
1997
Bidirectional RNN
Schuster & Paliwal
1997
LSTMs
Hochreiter & Schmidhuber
2006
Deep Belief Networks - pretraining
Hinton
2006
Deep Boltzmann Machines
Salakhutdinov & Hinton
2012
Dropout
Hinton
2014
GANs
Goodfellow
2017
Capsule Networks
Sabour, Frosst, Hinton
Made by Favio Vázquez

Real world problems

Audio Features

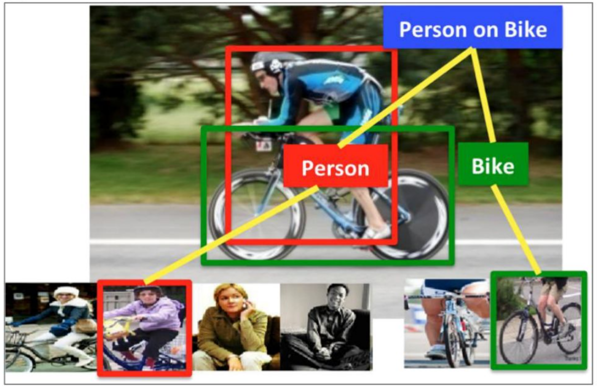


Spectrogram



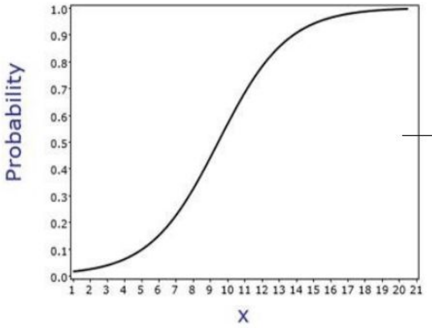
MFCC

- Object detection
- Action classification
- Image captioning
- …



"man in black shirt is playing guitar."

Logistic regression

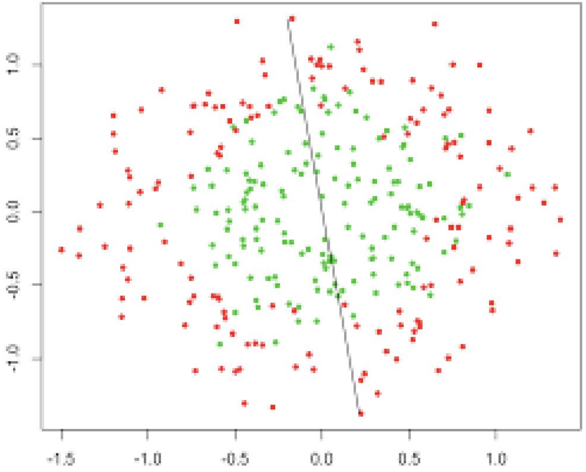
X → Wx +b → 

→ P(y)

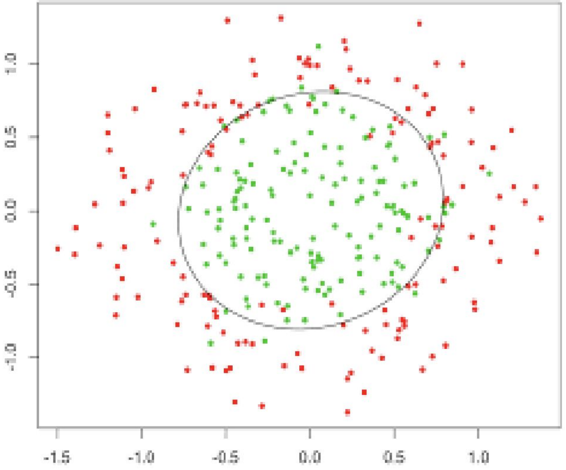
$$P(y|x) = \sigma(w \cdot x + b)$$

$$L = -\sum_i y_i \log P(y|x_i) + (1 - y_i) \log(1 - P(y|x_i))$$

Problem: nonlinear dependencies



What we have

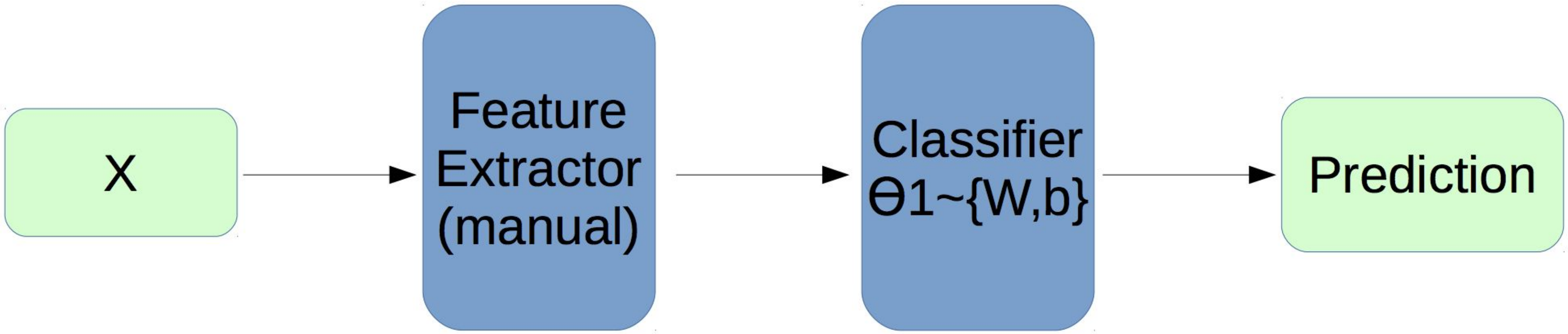


What we want

Logistic regression (generally, linear model) need feature engineering to show good results.

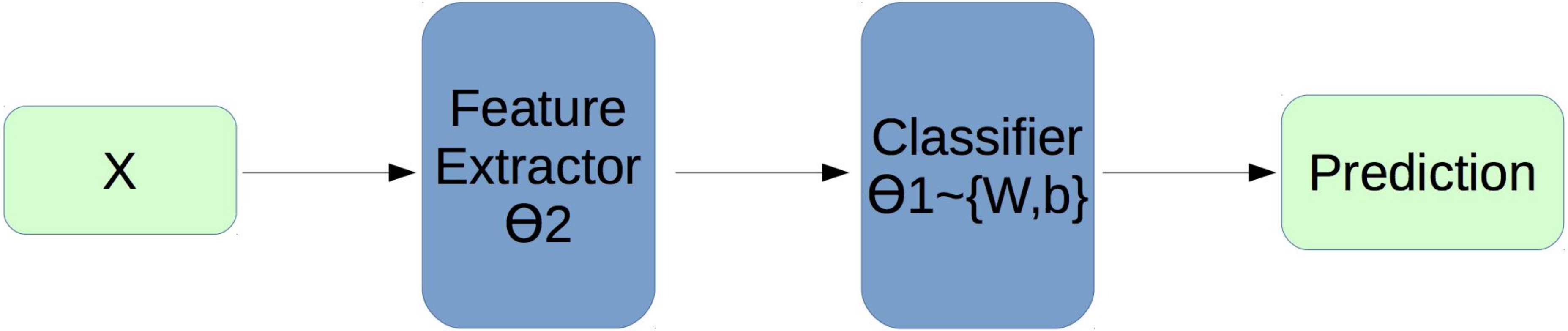
And feature engineering is an *art*.

Classic pipeline



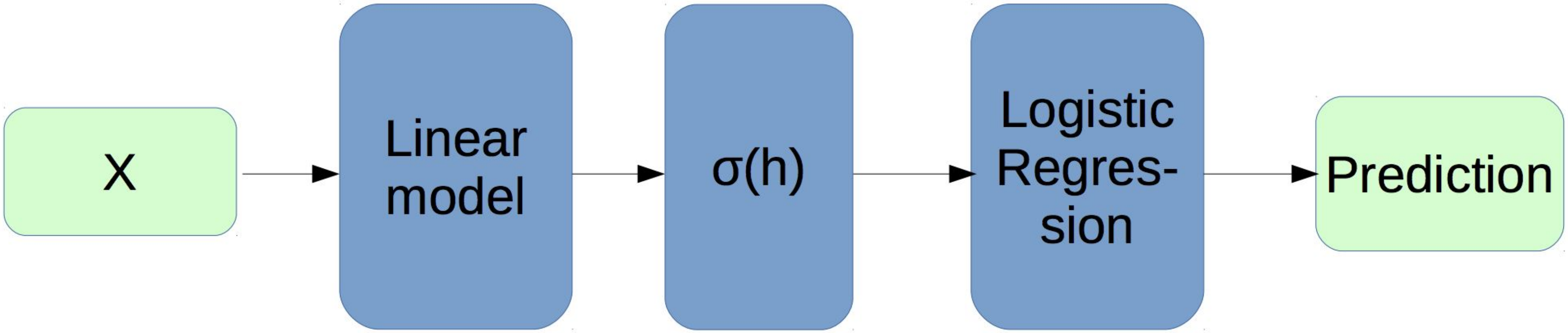
Handcrafted features, generated by experts.

NN pipeline



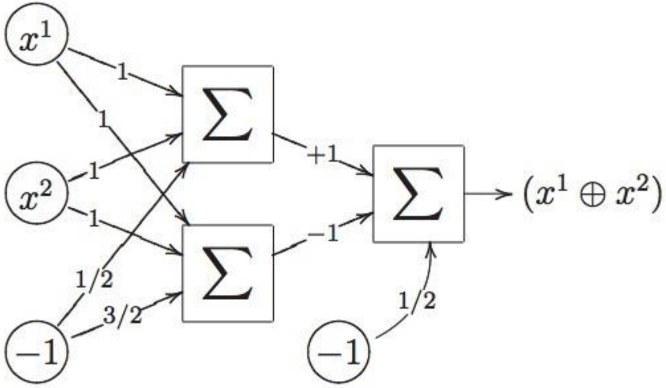
Automatically extracted features.

NN pipeline: example



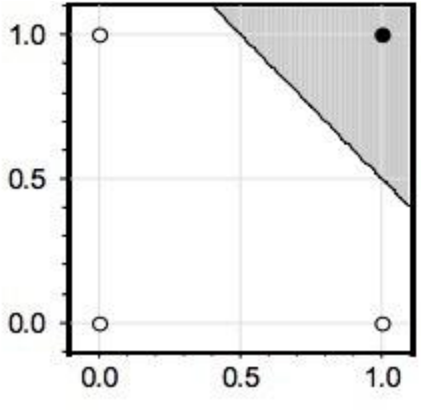
Actually, it’s a neural network.

XOR problem

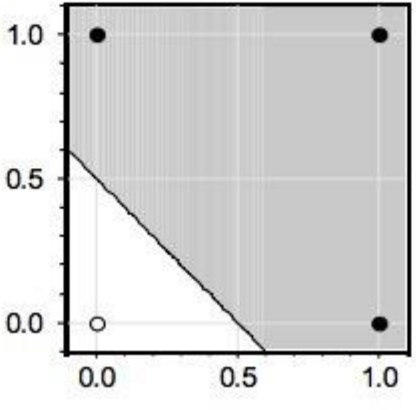


This 2-layer NN (on the left) implements XOR with only x1 and x2 features.

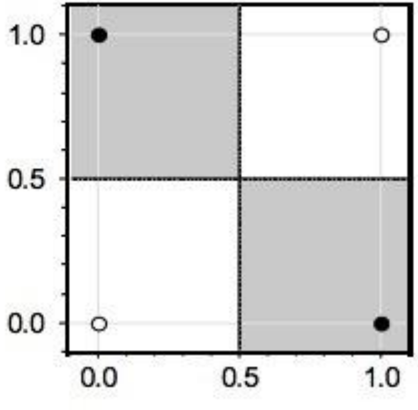
1-layer NN also can succeed, but only with extra feature x1*x2.



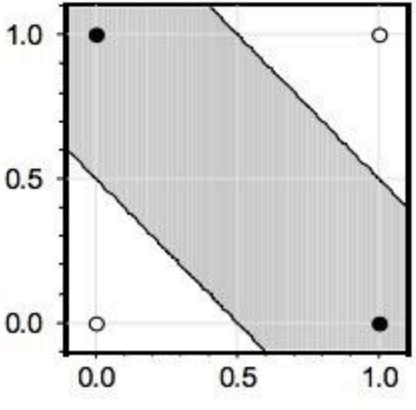
AND



OR

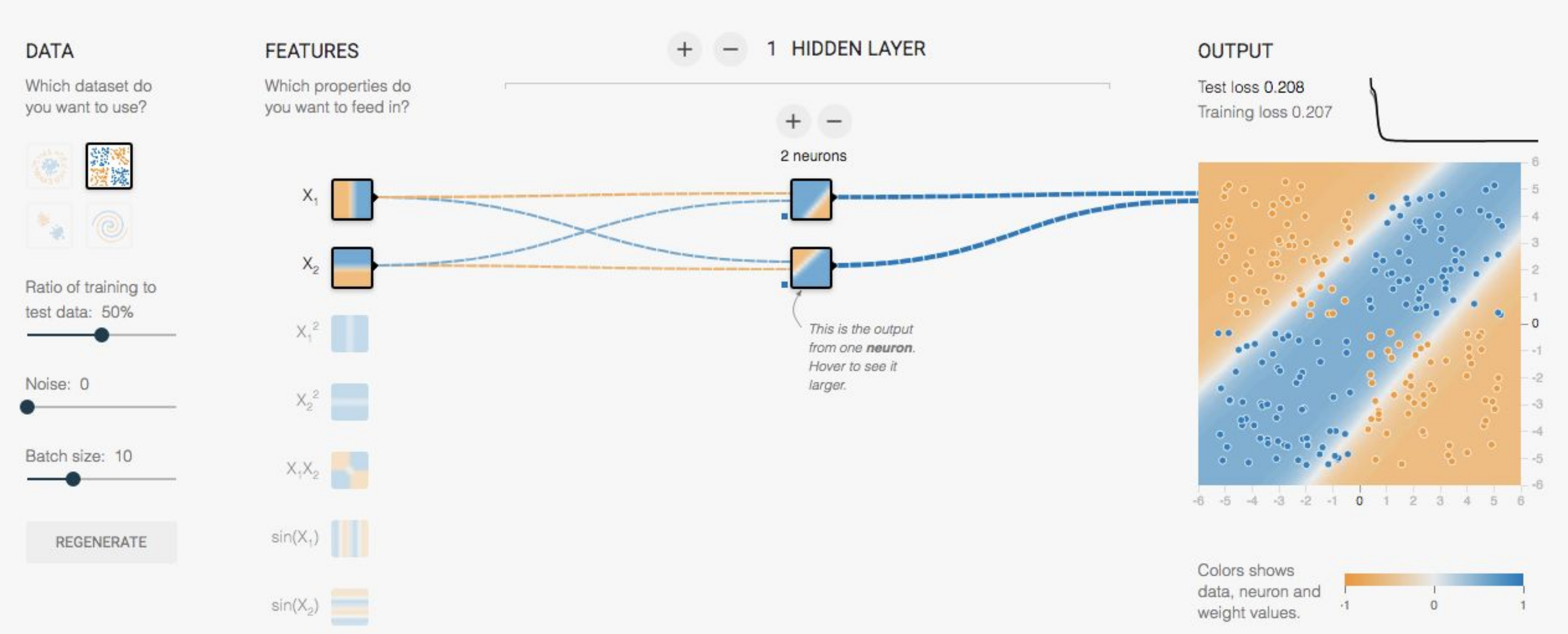


XOR(with x1*x2)



XOR

Practice time: interactive playground



http://bit.ly/ml4megafon_august18_NN_playground

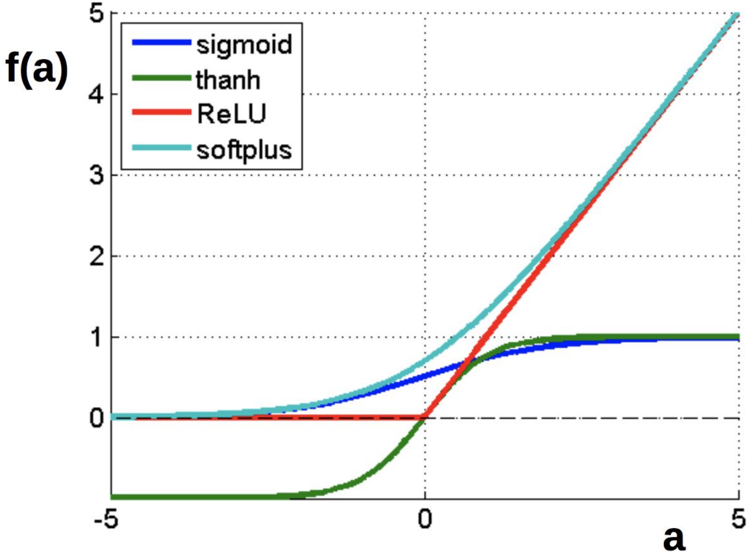
Once more: nonlinearities

$$f(a) = \frac{1}{1+e^a}$$

$$f(a) = \tanh(a)$$

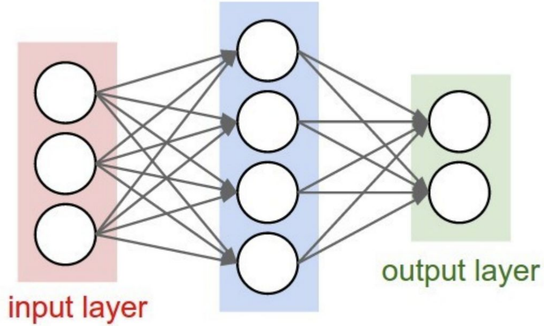
$$f(a) = \max(0, a)$$

$$f(a) = \log(1+e^a)$$



Some generally accepted terms

- Layer – a building block for NNs :
  - Dense layer: $f(x) = Wx+b$
  - Nonlinearity layer: $f(x) = \sigma(x)$
  - Input layer, output layer
  - A few more we gonna cover later
- Activation – layer output
  - i.e. some intermediate signal in the NN
- Backpropagation – a fancy word for “chain rule”

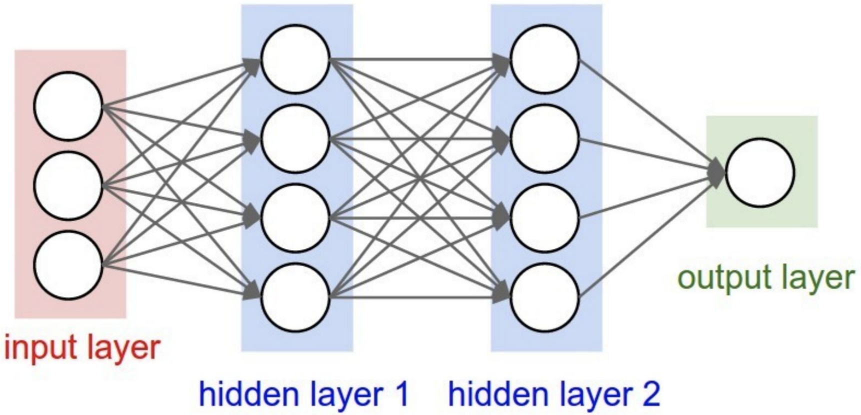


Dense layer
with a nonlinearity

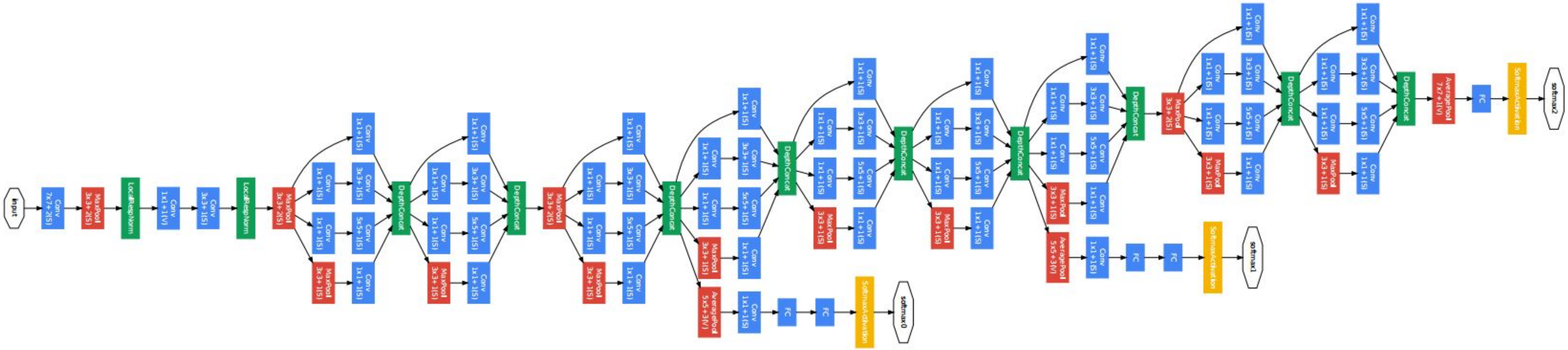
“Train it via backprop!”

BIGDATA TEAM

Actually, it can be deeper



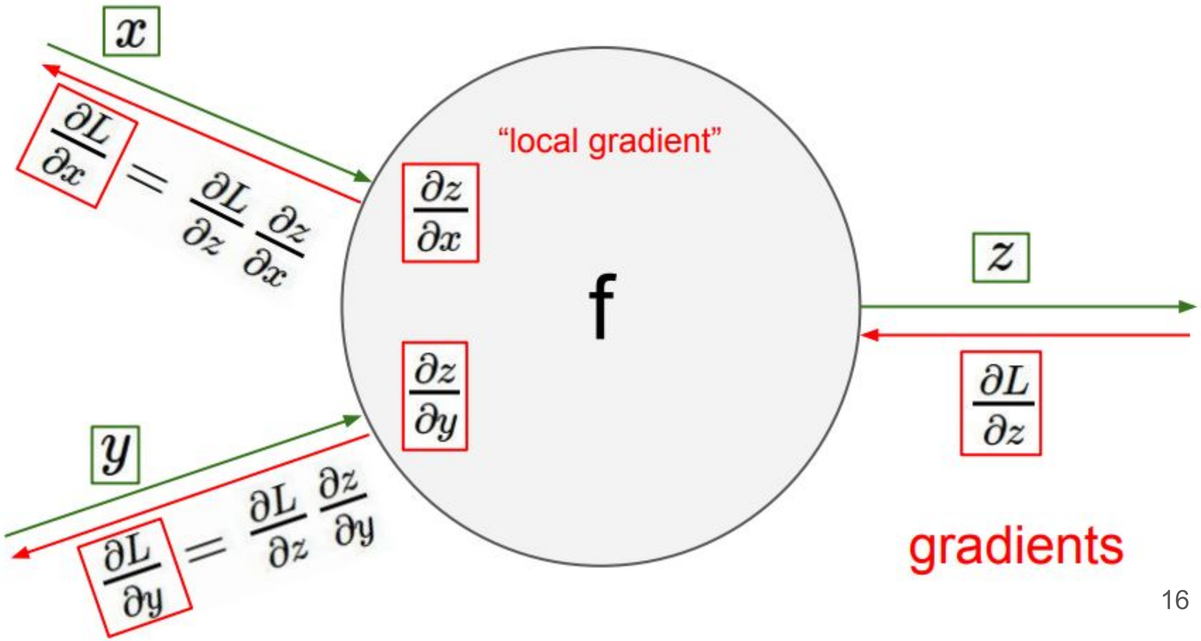
Much deeper...



Backpropagation and chain rule

Chain rule is just simple math: $$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

Backprop is just way to use it in NN training.

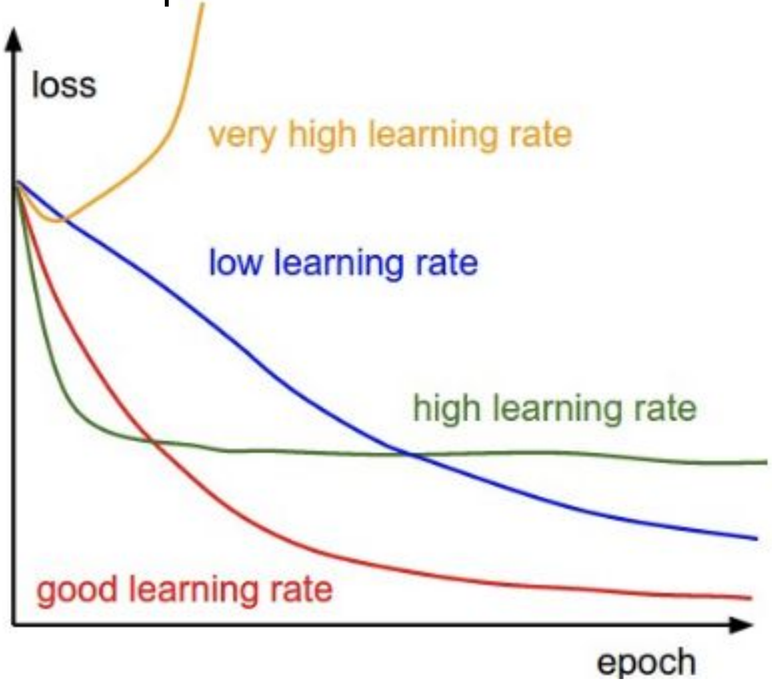


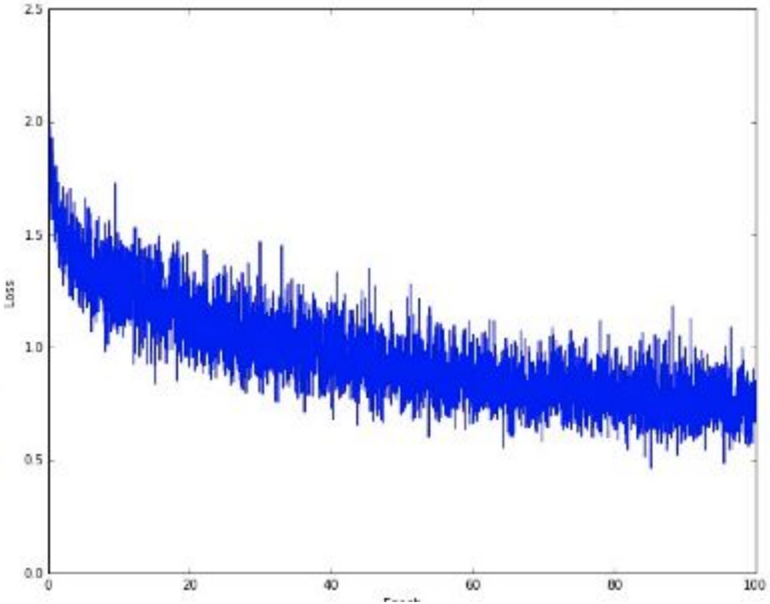
source: http://cs231n.github.io

Optimizers

Stochastic gradient descent is used to optimize NN parameters.

$$x_{t+1} = x_t - \text{learning rate} \cdot dx$$



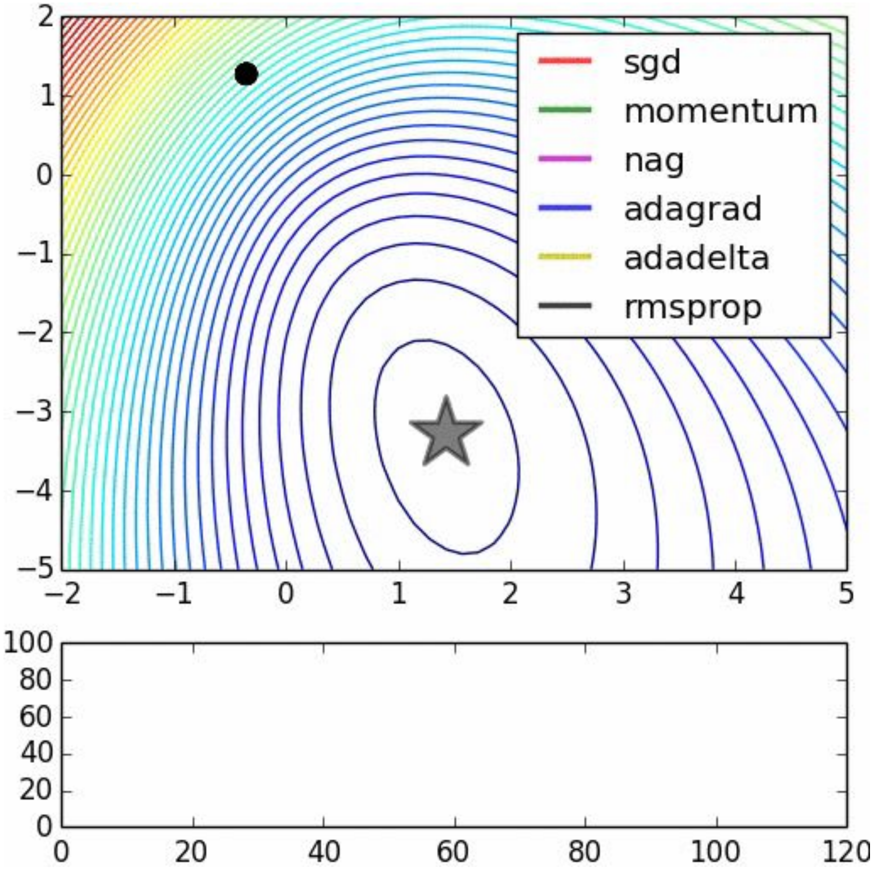


source: http://cs231n.github.io/neural-networks-3/

Optimizers comparison

There are much more optimizers:

- Momentum
- Adagrad
- Adadelta
- RMSprop
- Adam
- …
- even other NNs



Time to take a break

Leave your feedback, please:

http://bit.ly/ml4megafon_august18_lecture7_feedback

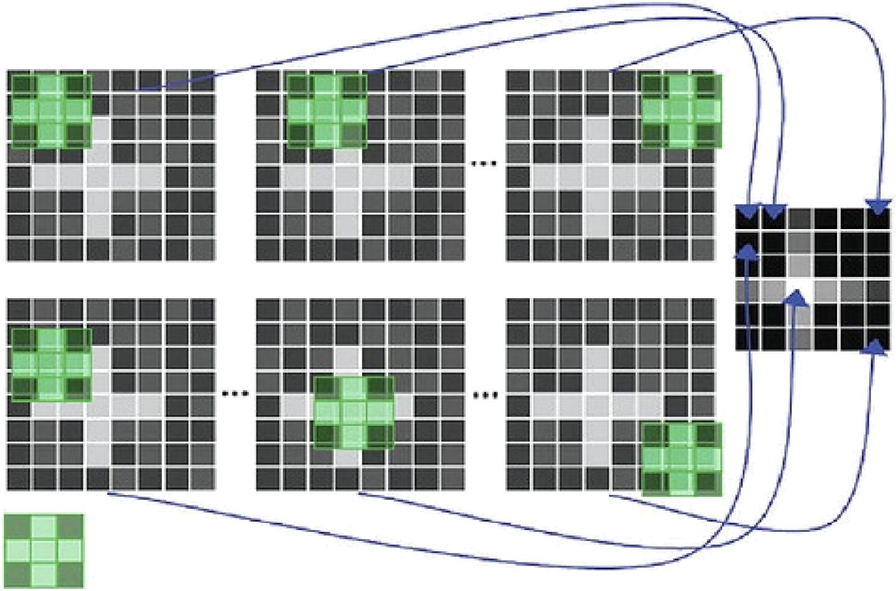
Back on track

Comes next:

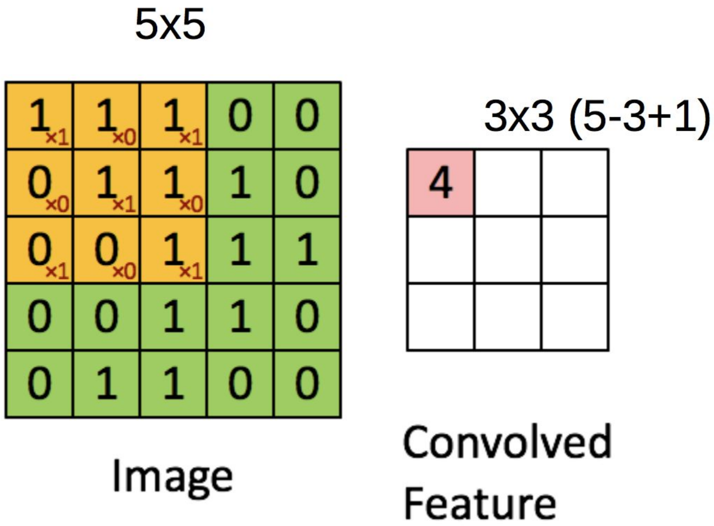
- More layers
  a. Convolutional layer
  b. Pooling layer
  c. Dropout layer
  d. Batchnorm layer (batch normalization)
  e. Embeddings (aka word2vec, GloVe)
  f. Recurrent ~~layer~~ neural networks
- More practice
  a. Names can’t wait to be generated by YOU

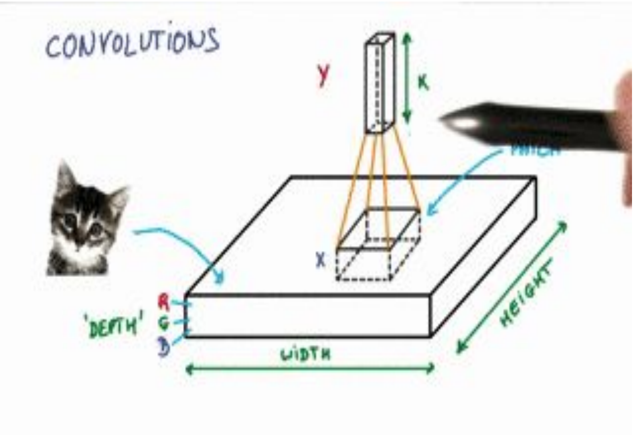
Convolution layer

Vital for Computer Vision and
Time Series Analysis problems



Convolution layer



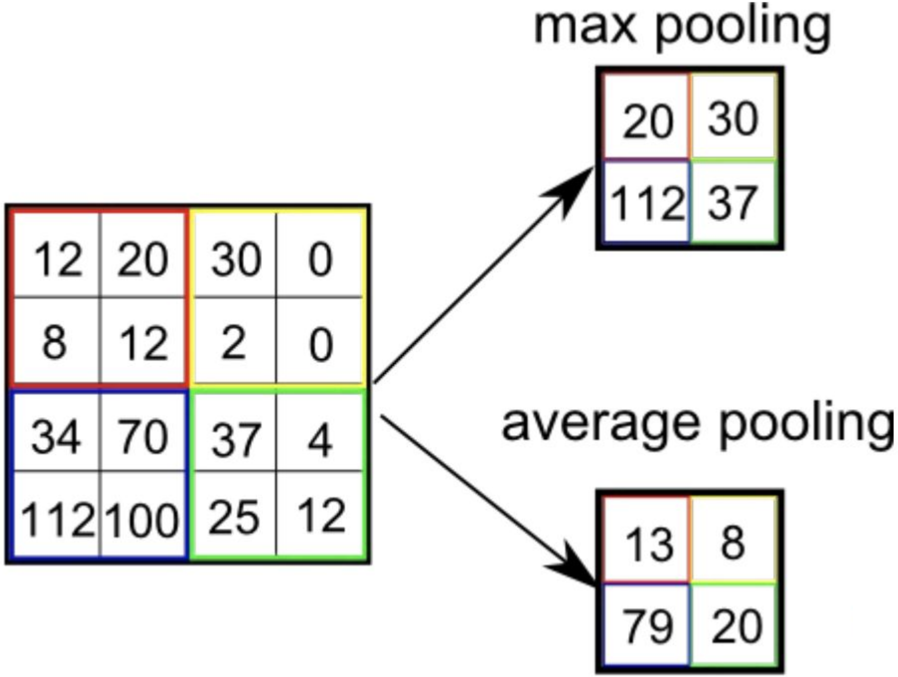


Intuition: how *cat-like* is this square?

Pooling layer

- Reduces layer size by a factor
- Makes NN less sensitive to small image shifts

- Widely used:
  - max pooling
  - mean pooling

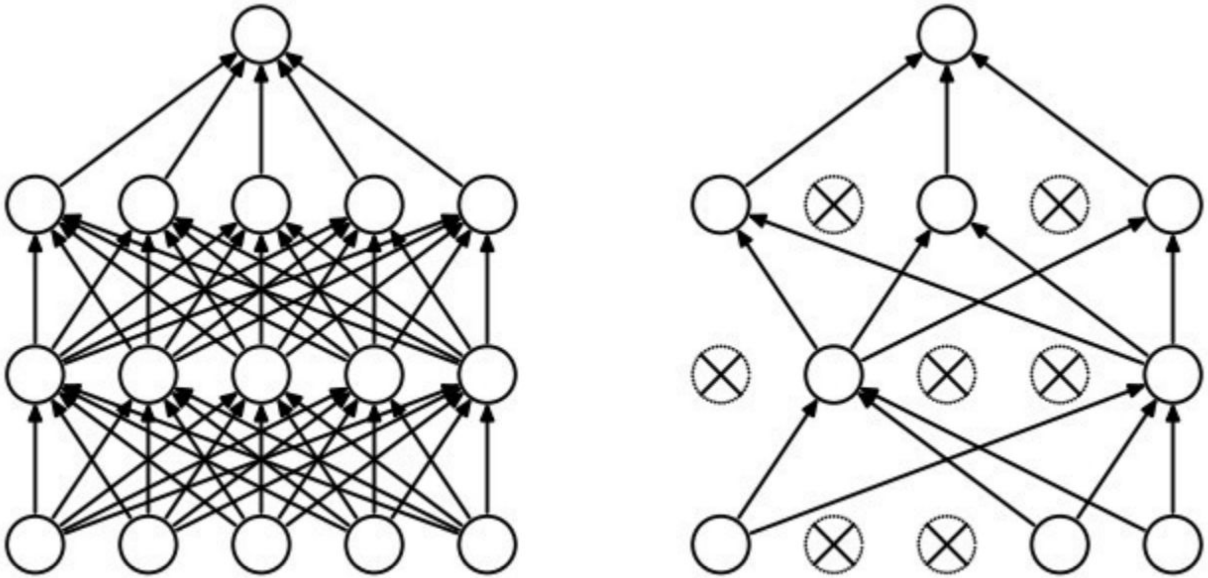


Dropout layer

Some neurons are “dropped” during training.

Prevents overfitting.

Actually, a form of regularization.



(a) Standard Neural Net (b) After applying dropout.

Batch normalization

Problem:

- Consider a neuron in any layer beyond first
- At each iteration we tune it's weights towards better loss function
- But we also tune it's inputs. Some of them become larger, some – smaller
- Now the neuron needs to be re-tuned for it's new inputs

Batch normalization

TL; DR:

- It's usually a good idea to normalize linear model inputs

(c) Every machine learning lecturer, ever

Batch normalization

- Normalize activation of a hidden layer

  (zero mean unit variance)

$$h_i = \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}$$

- Update μ_i, σ_i^2 with moving average while training

$$\mu_i := \alpha \cdot mean_{batch} + (1-\alpha) \cdot \mu_i$$

$$\sigma_i^2 := \alpha \cdot variance_{batch} + (1-\alpha) \cdot \sigma_i^2$$

But what about NLP?

- Bag of words
- TF-IDF
- Ensembles
- ...

BIGDATA TEAM

RNNs generating...

Shakespeare

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

Algebraic Geometry (Latex)

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*
Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

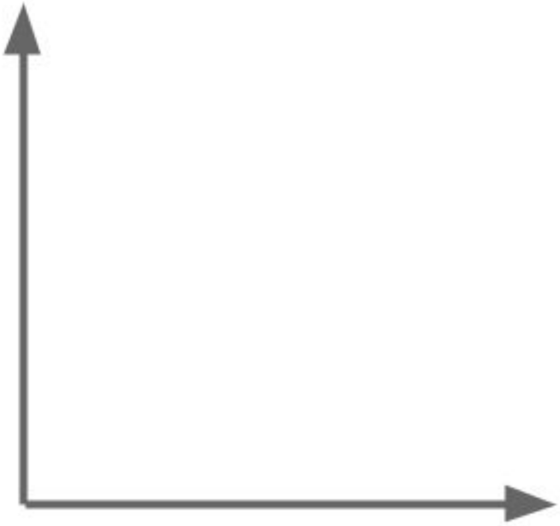
Linux kernel (source code)

```c
/*
 * If this error is set, we will need anything right after that BSD.
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long) *FIRST_COMPAT);
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  div_u64_w(val, inb_p);
  spin_unlock(&disk->queue_lock);
  mutex_unlock(&s->sock->mutex);
  mutex_unlock(&func->mutex);
  return disassemble(info->pending_bh);
}
```

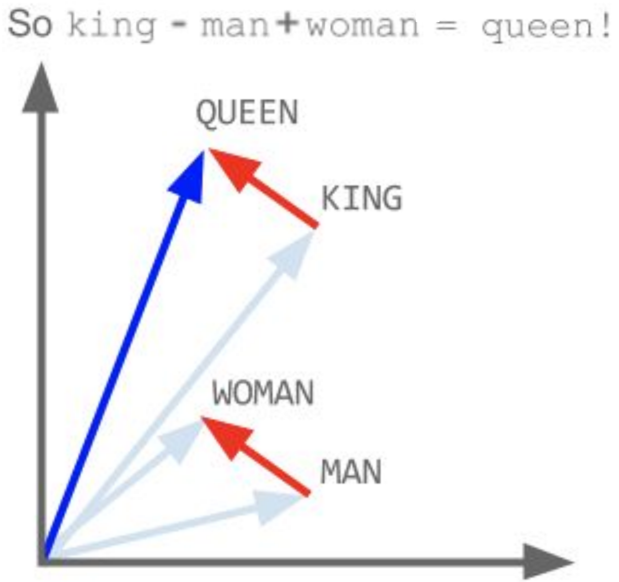
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Embeddings: intuition

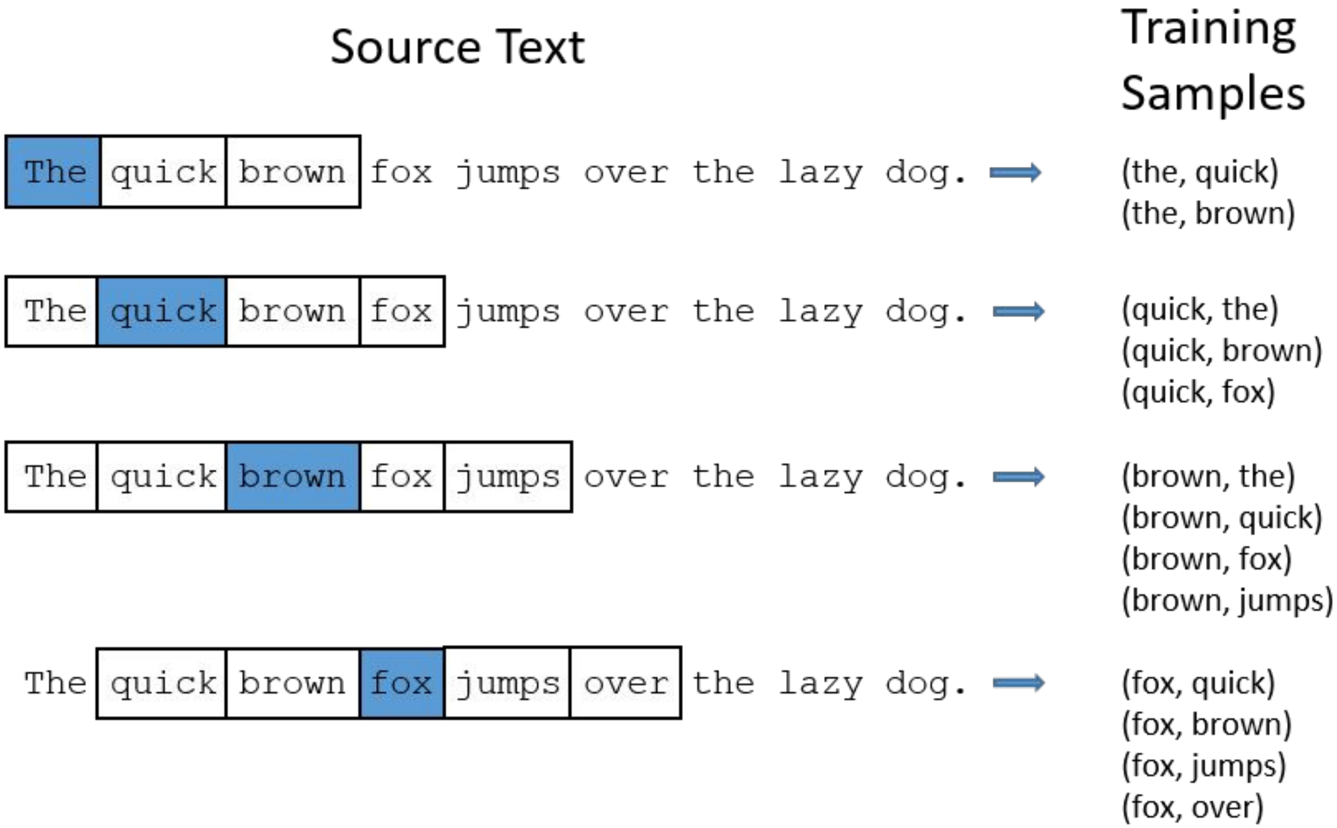
What is king - man + woman?



Embeddings: intuition

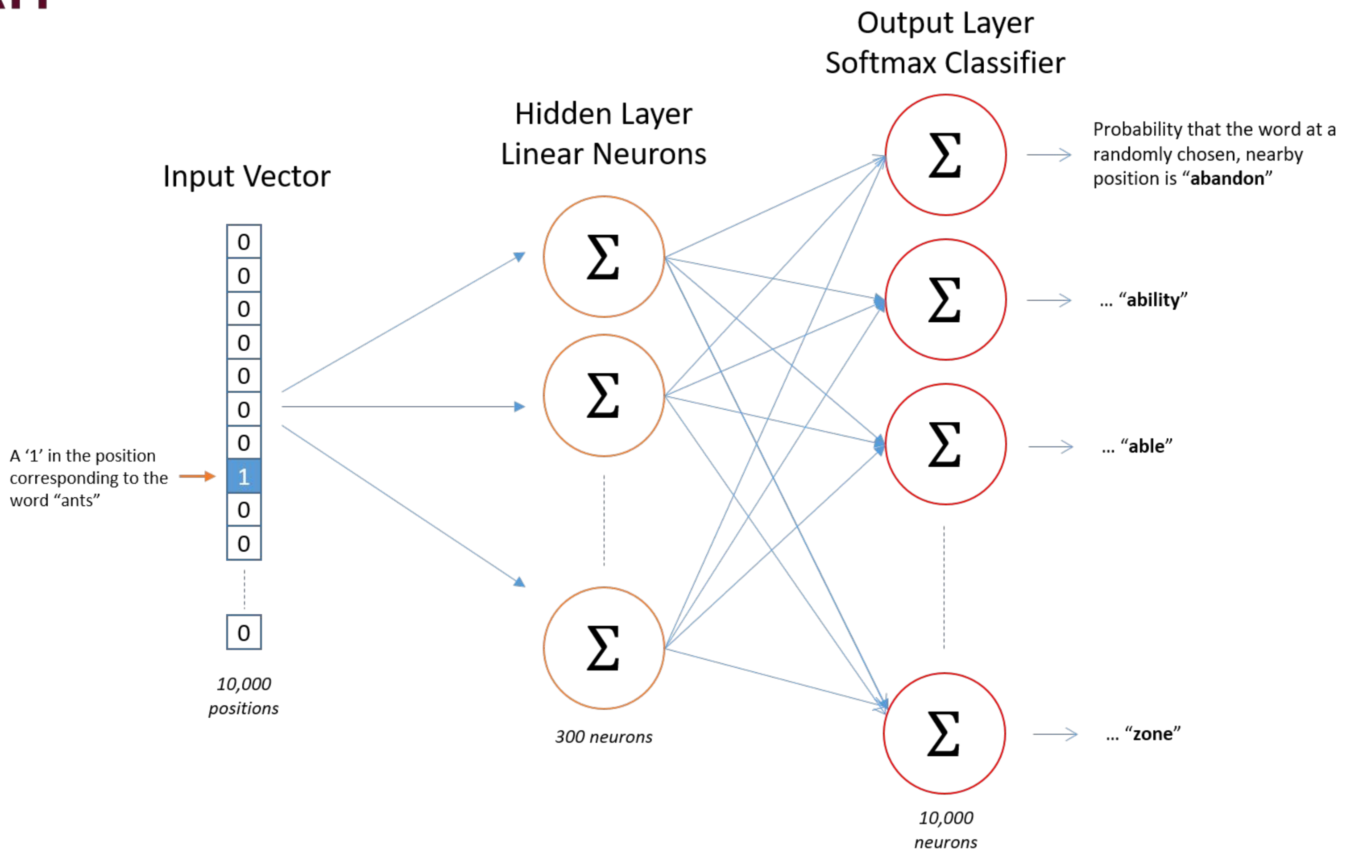


Embeddings: word2vec



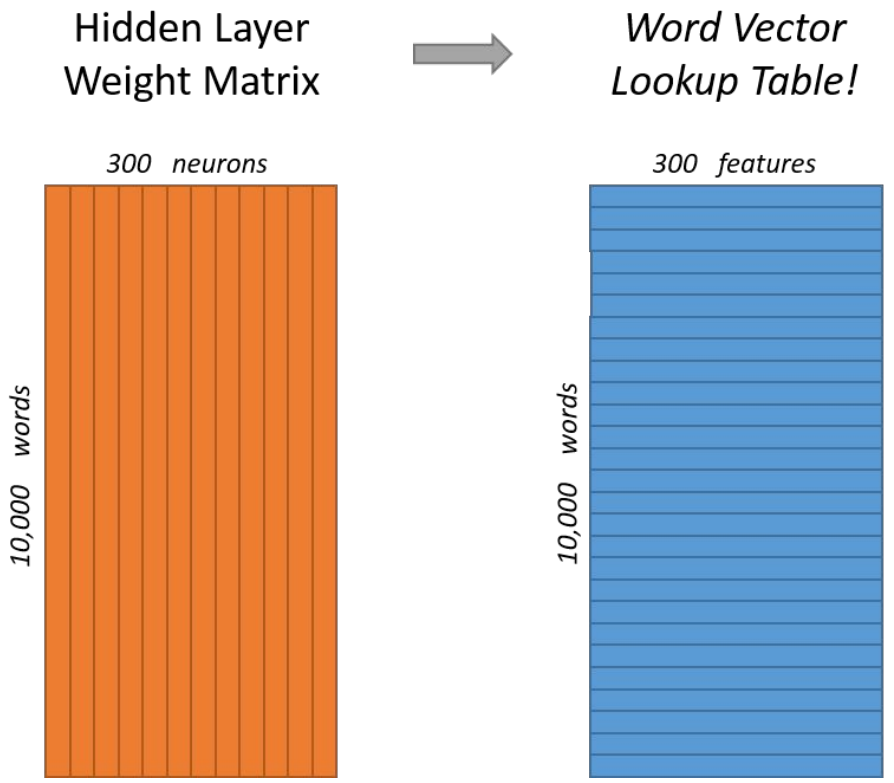
Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec



Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec



Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec

- Word vectors with 300 components
- Vocabulary of 10,000 words.
- Weight matrix with 300 x 10,000 = 3 million weights each!

Training is too long and computationally expensive

How to fix this?

Embeddings: word2vec

Basic approaches:

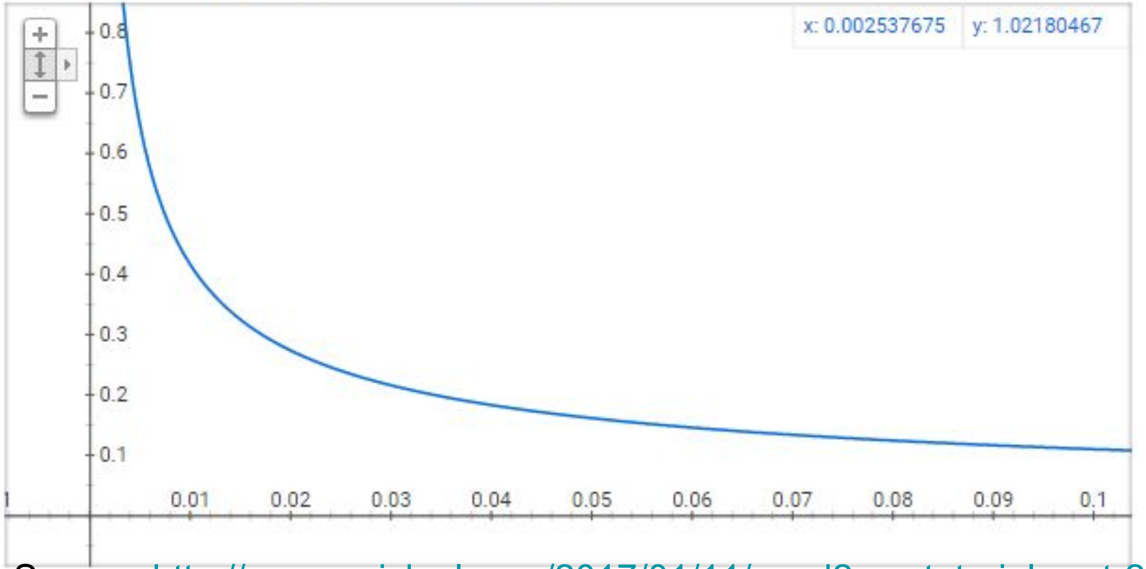
1. Treating common word pairs or phrases as single “words” in their model.
2. Subsampling frequent words to decrease the number of training examples.
3. Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update only a small percentage of the model’s weights.

Embeddings: word2vec

Subsampling frequent words.

w_i is the word, $z(w_i)$ is the fraction of this word in the whole text,

Graph for (sqrt(x/0.001)+1)*0.001/x



$P(w_i)$ is the probability of *keeping* the word:

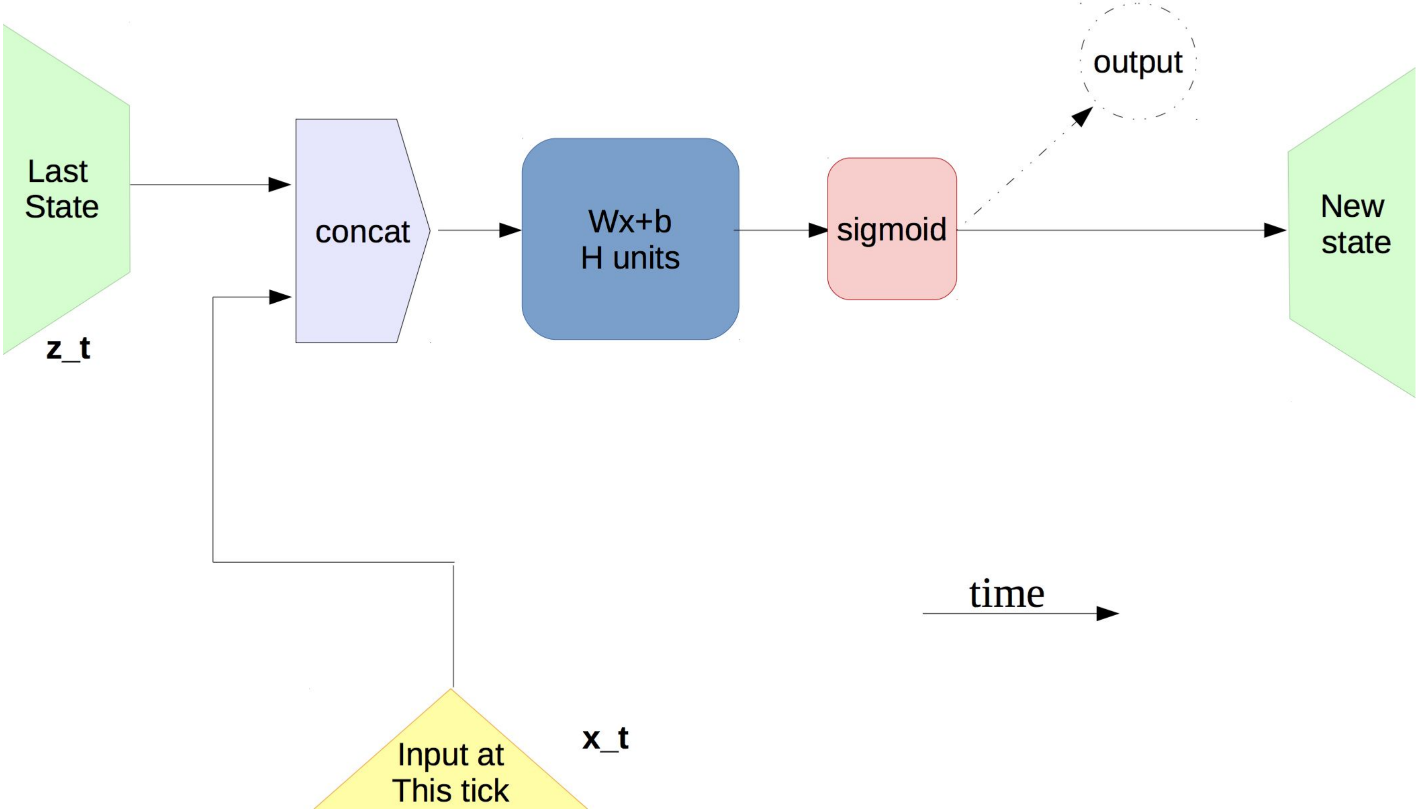
$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

Source: http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

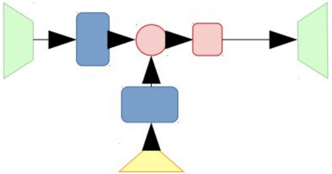
Embeddings: word2vec

Negative Sampling idea: only few words error is computed. All other words has zero error, so no updates by the backprop mechanism.

Recurrent neural network

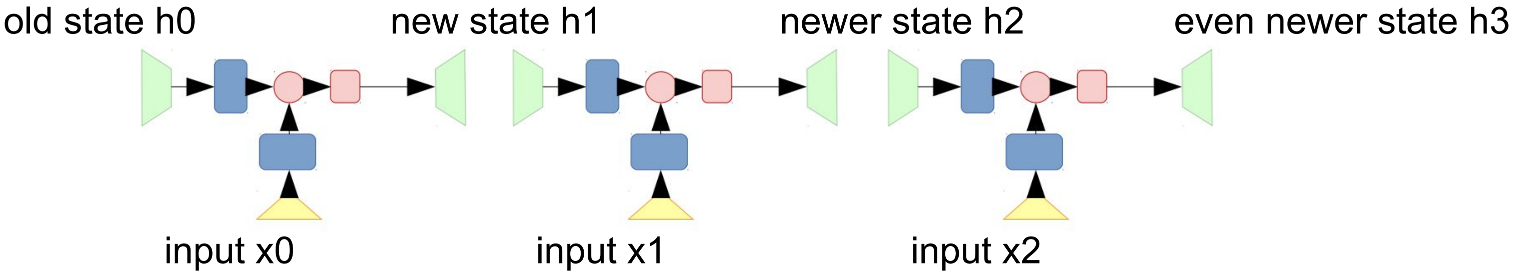


Recurrent neural network

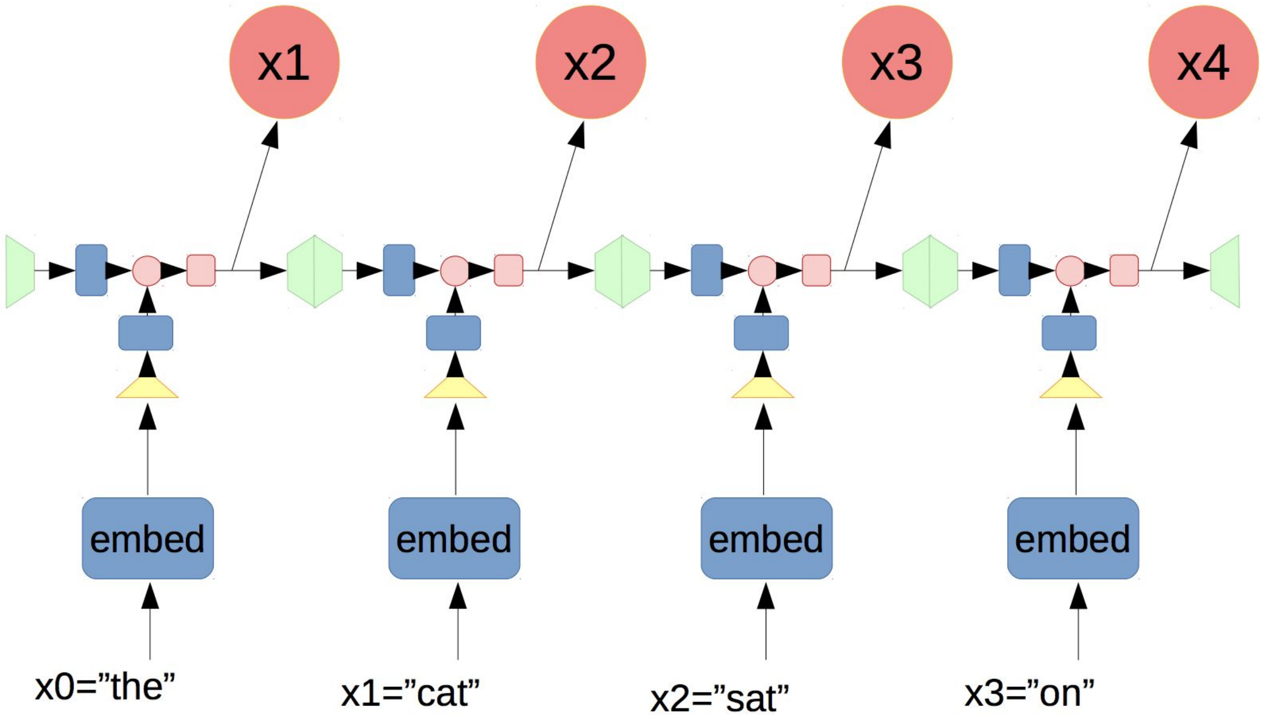


Recurrent neural network

We use same weight matrices for all steps



Recurrent neural network



Recurrent neural network

Now with formulas

$$h_0 = \bar{0}$$

$$h_1 = \sigma\big(\langle W_{\text{hid}}[h_0, x_0]\rangle + b\big)$$

$$h_2 = \sigma\big(\langle W_{\text{hid}}[h_1, x_1]\rangle + b\big) = \sigma\big(\langle W_{\text{hid}}[\sigma\big(\langle W_{\text{hid}}[h_0, x_0]\rangle + b, x_1]\rangle + b\big)$$

$$h_{i+1} = \sigma\big(\langle W_{\text{hid}}[h_i, x_i]\rangle + b\big)$$

$$P(x_{i+1}) = \text{softmax}\big(\langle W_{\text{out}}, h_i\rangle + b_{\text{out}}\big)$$

BIGDATA TEAM

That’s all, RNNs, we’re coming

But first, feedback, please:

http://bit.ly/ml4megafon_august18_lecture8_feedback